Scalable Reed-Solomon-based Reliable Local Storage for HPC Applications on IaaS Clouds

Leonardo Bautista Gomez, Bogdan Nicolae, Naoya Maruyama, Franck Cappello, Satoshi Matsuoka

Scalable Reed-Solomon-based Reliable Local Storage for HPC Applications on IaaS Clouds

Leonardo Bautista Gomez[1], Bogdan Nicolae[2], Naoya Maruyama[5], Franck Cappello[2,3], and Satoshi Matsuoka[1,4]

[1] Tokyo Institute of Technology, Japan
[2] INRIA, France
[3] University of Illinois at Urbana Champaign, USA
[4] National Institute of Informatics, Japan
[5] RIKEN AICS, Japan

Abstract. With increasing interest among mainstream users to run HPC applications, Infrastructure-as-a-Service (IaaS) cloud computing platforms represent a viable alternative to the acquisition and maintenance of expensive hardware, often out of the financial capabilities of such users. Also, one of the critical needs of HPC applications is an efficient, scalable and persistent storage. Unfortunately, storage options proposed by cloud providers are not standardized and typically use a different access model. In this context, the local disks on the compute nodes can be used to save large data sets such as the data generated by Checkpoint-Restart (CR). This local storage offers high throughput and scalability but it needs to be combined with persistency techniques, such as block replication or erasure codes. One of the main challenges that such techniques face is to minimize the overhead of performance and I/O resource utilization (i.e., storage space and bandwidth), while at the same time guaranteeing high reliability of the saved data. This paper introduces a novel persistency technique that leverages Reed-Solomon (RS) encoding to save data in a reliable fashion. Compared to traditional approaches that rely on block replication, we demonstrate about 50% higher throughput while reducing network bandwidth and storage utilization by a factor of 2 for the same targeted reliability level. This is achieved both by modeling and real life experimentation on hundreds of nodes.

1 Introduction

In recent years High Performance Computing (HPC) applications have seen an increasing adoption among mainstream users, both in academia and industry. Unlike "hero" applications that are designed to run on powerful (and expensive!) supercomputers, mainstream users typically need to run medium-sized jobs that need no more than a couple of thousands of cores. For these types of jobs, Infrastructure-as-a-Service (IaaS) [4] cloud platforms present a viable alternative to purchasing dedicated resources: with thousands of virtual machines (VMs) routinely allocated by large IaaS providers [2], users can easily lease a virtual environment on the cloud for their HPC applications.

However, running HPC applications in an efficient fashion on IaaS clouds is challenging. One such open challenge is how to handle storage. Unlike supercomputing infrastructure, where storage is typically handled using a POSIX-compatible parallel file

system (e.g., GPFS [22] or PVFS [10]), IaaS clouds feature a large variety of specialized storage solutions that are not standardized, which makes it difficult to port HPC applications. Furthermore, these storage services are often geared towards high-availability rather than high performance, not to mention that they incur costs proportional to the I/O space and bandwidth utilization. One solution to this problem is to rely on the local storage available to the VM instances. In a common IaaS configuration, local storage is plentiful (several hundreds of GB), up to an order of magnitude faster [17] and does not incur any extra operational costs. Furthermore, most HPC applications can directly take advantage of local storage or require little modifications to do so, which greatly increases scalability.

Despite its obvious advantages, local storage has a major issue: it relies on commodity components that are prone to failures [24]. Even if local disks did not fail, they would become inaccessible if their hosting compute nodes failed, effectively leading to loss of data. As a consequence, we need to deal with the reliability of local storage in order to be able to leverage it in our context. However, this invariably introduces an additional overhead, both performance-wise and resource-wise. Current cloud storage services achieve reliability and availability by replicating data, often three or more copies. However, data replication is highly space and bandwidth consuming, and it leads to an inefficient use of available resources. In this paper we explore the use of Reed-Solomon (RS) [21] based erasure encoding to address the reliability requirement for local storage in a scalable and efficient fashion. We aim to achieve a low overhead for our scheme, such that it can sustain a high I/O data access throughput and a high reliability level with minimal storage space and bandwidth utilization.

Our contributions can be summarized as follows:

- We propose a novel Reed-Solomon based encoding algorithm specifically optimized to conserve total system bandwidth in scenarios where large amounts of data are concurrently dumped to the local disks, which ultimately diminishes operational costs and frees up more bandwidth for the applications themselves. (Section 3.1)
- We introduce a formal model to compare data-replication and RS encoding analytically in order to predict the storage space and network bandwidth utilization for different levels of reliability. (Section 4)
- We show how to implement our approach in practice by integrating it into *BlobCR* [19], a distributed checkpoint-restart framework that is specifically designed to take persistent snapshots of local storage for HPC applications that are running on IaaS clouds. (Section 3.2)
- We evaluate our approach experimentally on hundreds of nodes of the Grid'5000 testbed [7], both with synthetic benchmarks and a real-life application. These experiments demonstrate significant improvement in performance and resource utilization when compared to replication for the same reliability level. (Section 5)

2 Related work

There is a rich storage ecosystem around IaaS clouds. Cloud providers typically offer their own storage solutions, which are not standardized and expose a different access

model than POSIX: key-value stores based on REST-ful access APIs [3], distributed file systems specialized for MapReduce workloads [23], database management systems [14], etc. Besides the disadvantages presented in Section 1, most of these solutions are optimized for high-availability, under the assumption that data is frequently read and only seldom updated. HPC applications on the other hand typically generate new data often and need to read it back only rarely (e.g. checkpointing). The idea of leveraging local storage for HPC applications running on IaaS clouds in a reliable fashion was introduced before by our previous work: we presented *BlobCR* [19], a checkpoint-restart framework that is able to take snapshots of local storage that survive failures. However, in order to survive failures, BlobCR relies on replication, which can lead to excessive use of storage space and bandwidth.

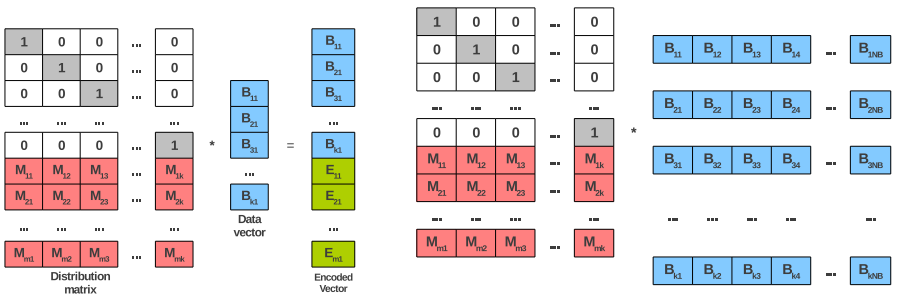
Different studies of block replication and erasure codes were performed before in the context of RAID systems [8], whose implementation is at hardware level, as well as for distributed data storage [25] implemented at the software level. More recently, DiskReduce [12] and Zhe Zhang et al. [26] study the feasibility of replacing three-way replication with erasure codes in cloud storage / large data centers. They are probably the closest work to the technique proposed in this paper, going in a similar direction and complementing our own approach: while they focus on space reduction and performance, in this work we focus on limiting the network bandwidth consumption as much as possible by using a novel low-communication RS encoding, which significantly decreases the application performance overhead.

RS encoding algorithms [15, 11] similar to the proposal presented in our previous work create encoding groups that encompass blocks stored on different nodes. Other encoding techniques, such as bitwise XOR [16] also encode distributed blocks. Such algorithms are suitable for scenarios such as coordinated checkpoint where multiple distant processes need to reliably store data at the same time; in such context, distributed blocks of data can be encoded after synchronization. However, these algorithms cannot be used for a storage system where isolated writes will need to be performed without imposing any synchronization with other processes. Therefore, a fundamental algorithm change is necessary in order to encode isolated blocks of data.

3 Our approach: a RS-encoding algorithm proposal

In this section we introduce a novel low-communication RS encoding algorithm that guarantees high reliability with a low bandwidth consumption. The key observation leading to this choice is the widening gap between the cost of computational power and network bandwidth, for which reason we try to conserve network bandwidth at the expense of slightly higher CPU utilization.

We start with a quick overview of the RS encoding. As we can see in Figure 1(a), the RS encoding takes a data vector and encodes it by performing a matrix-vector multiplication with a distribution matrix. To recover any m failures, any sub-square matrix of the distribution matrix (including minor) must be non-singular. Thus, in practice we usually use a Cauchy or a Vandermonde matrix [20]. While using RS encoding to encode distributed data, the data vector is composed by blocks of data of distant nodes. In Figure 1(b), we can see the data to be encoded in the case of diskless checkpoint-



(a) Reed-Solomon encoding (b) Reed-Solomon for distributed data

Fig. 1. Reed-Solomon encoding study

ing. Each process P_i holds blocks of data B_{ij} where j is the block index, going from 1 to NB (Number of Blocks in the checkpoint file). The first data vector will be composed of the first block of each checkpoint file, the second vector of the second block of each checkpoint file, and so on. There are several ways to implement this computation, for example, one can use MPI reductions [15], a pipeline algorithm [11] or the star algorithm proposed in our previous work [13]. However, all these algorithms require synchronizations while encoding the data. Furthermore, these algorithms produce as much (or more) communications than data replication; therefore, none of them is suitable for our purpose.

3.1 Low-communication RS encoding algorithm

In an ideal setting where all m encodings generated by RS are stored on different nodes (assuming that the data blocks themselves are stored on different nodes as well), the system can tolerate m simultaneous erasures. However, this ideal scenario is costly in terms of resource utilization, as it results in the need to employ dedicated parity nodes. Thus, to leverage the available resources better, one idea is to store the encodings on the same nodes where the data itself is stored. The distributed algorithm (Algorithm 1) presented in our previous work [13, 5] illustrates this idea. We propose to generate as many encodings as the group size ($m = k$), while evenly distributing the encodings among the same nodes where the data blocks are stored. Thus, one node failure will lead to two erasures and each group will be able to tolerate up to 50% of failed nodes.

In order to avoid synchronizations between processes during the encoding, we propose a novel low-communication algorithm (Algorithm 2). Instead of encoding the i^{th} block of each one of the k nodes, we encode the first k blocks of each node and then we scatter the k blocks of original data plus the k encodings on the k nodes of the group. Notice that both algorithms encode the same amount of data and store all the original and parity data on the encoding nodes, thus offering the same level of reliability.

Algorithm 1 Distributed RS encoding algorithm

```
1:                                  ▷ r : the process rank
2:                          ▷ NB : the number of blocks
3:                                  ▷ k : the group size
4:                       ▷ m : the number of encodings
5:  for i ← 1..NB do
6:      B_ri ← read i^th block
7:      for j ← 1..m do
8:          T ← M_(r+j)r * B_ri
9:          Send T to P_r+j
10:         F ← Recv from P_r−j
11:         E_ri ← E_ri + F
12:     end for
13:     writeE_ri
14: end for
```

Algorithm 2 Low-communication RS encoding algorithm

```
1:  for i ← 0..(NB/k) − 1 do
2:      for j ← 1..k do
3:          B_r(i*k+j) ← read (i*k + j)^th block
4:      end for
5:      for j ← 1..k do
6:          for l ← 1..m do
7:              E_ij ← E_ij + M_jl * B_r(i*k+l)
8:          end for
9:      end for
10:     for j ← 1..k do
11:         Send B_r(i*k+j+r) and E_i(j+r) to P_j+r
12:         Recv FB_j and FE_j from P_j−r
13:         Write FB_j and FE_j
14:     end for
15: end for
```

With respect to bandwidth consumption, Algorithm 1 needs to transfer $NB * m$ blocks in total. Since $m = k$, its communication cost is thus $Comm_{Alg_1} = NB * k$. On the other hand, for Algorithm 2 we have $Comm_{Alg_2} = \frac{NB}{k} * k * 2 = NB * 2$. By comparing the two formulas one can easily notice that the low-communication algorithm is more bandwidth friendly than the distributed algorithm. Particularly, in the case of the distributed algorithm the data transferred over the network will increase proportionally with the group size k, while the low-communication algorithm will keep it constant.

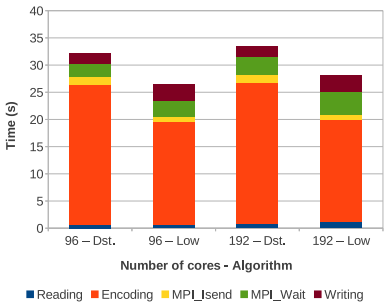
Figure 2(a) shows a performance comparison and time breakdown of both encoding algorithms measured on Tsubame2 [1], for different numbers of cores (96 and 192 cores). Although communications and computation are overlapped in both cases, the low-communication algorithm is 24% faster because of the data locality. Reducing communications not only decreases the stress on the network but also increases cache efficiency. For isolated writes on a storage system, one can design a system that implements a multi-stage striping: each chunk of data that makes up the local storage can be itself divided into the k blocks that are fed to the low-communication encoding algorithm.

3.2 Integration in practice

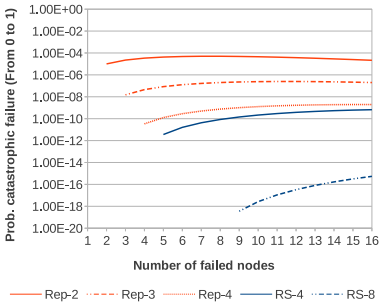
In order to illustrate the benefits of the algorithm presented in the previous section in practice, we have integrated our approach into *BlobCR* [19], a distributed checkpoint-restart framework based on *BlobSeer* [18] that is specifically designed to take persistent snapshots of local storage for HPC applications that are running on IaaS clouds. BlobCR exposes local storage to the VM instances as virtual disks that can be attached to them. The initial content of the virtual disk is striped into chunks and stored in a distributed fashion among the participating storage elements. Whenever a virtual disk is attached to a VM instance, an initially empty mirror of it is created on the local disk.

Reads to the virtual disk fetch any remote chunks not present in the mirror, gradually filling it on-demand. Writes to the virtual disk are always performed on the mirror. A special primitive can be used to persistently save the mirror as a new snapshot of the virtual disk that is globally shared. This is done by distributing all locally modified chunks among the storage elements, then by consolidating these changes using cloning and shadowing.

In order to provide high reliability for the chunks that make up the disk snapshots, BlobCR relies on the reliability scheme implemented in BlobSeer, which by default is replication: each chunk is stored to multiple local disks. We implemented an alternative reliability scheme based on the algorithm presented in Section 3.1 which was then integrated into BlobSeer. More precisely, instead of replicating each chunk to multiple local disks, we perform a second level of striping that splits each chunk into k small, equally sized blocks. These blocks form a group to which erasure coding is applied in order to obtain a second group of k blocks that hold parity information. Once this step has completed, we distribute the $2*k$ blocks among a set of $2*k$ different remote disks.



(a) Performance of Algorithm 1 vs. Algorithm 2 (1 GB/core)



(b) Reliability of replication vs. RS encoding (1000 nodes)

Fig. 2. Performance compared to previous work and reliability modeling

4 Reliability, storage and network bandwidth study

In this section we develop a model to predict the performance, storage and network bandwidth cost of both approaches (data replication and RS encoding) for comparable levels of reliability. As explained in Section 3, the distributed encoding algorithm is not suitable for storage systems, therefore we do not include it in this comparison.

First, we focus on the reliability level of both approaches. We use the reliability model presented in our previous work [6] to compute the probability of *catastrophic failures*, i.e., failures that lead to unrecoverable data loss. This will depend on the number of simultaneous erasures and the probability of those erasures to hit the replicated

or parity data of a given data chunk. Figure 2(b) shows the difference of reliability for five different settings: our approach using a group size of four and eight (denoted *rs-4* and *rs-8*) vs. replication using a replication factor of two, three and four (denoted *rep-2*, *rep-3* and *rep-4* respectively). Notice that *rep-4* comes very close (without surpassing) to *rs-4* in terms of reliability only when the replication factor reaches 4. For this reason, we consider *rep-4* and *rs-4* comparable in terms of reliability for the rest of this paper.

A fair comparison between both techniques should study the storage overhead and performance overhead necessary to guarantee a comparable level of reliability. The replication technique replicates chunks of data and stores them on multiple remote disks. Similarly, the RS encoding technique generates parity data and stores it on multiple remote nodes. Let us assume that we want to reliably write a chunk of z bytes of data. We assume a replication factor of k and an RS encoding group size of k. In the replication approach we store a total of $Str_{rep} = z * k$ bytes of data. In contrast, for the RS technique we need to split the chunk in k blocks, encode them and finally store a total of $Str_{rs} = k * \frac{z}{k} * 2 = z * 2$ bytes in the system. As we can see, the replication approach becomes prohibitively expensive quickly, while RS encoding is scalable in terms of storage. In addition, the amount of data transferred over the network is equal to the amount of data stored for both approaches, which means that data replication transfers more data than RS encoding for a similar level of reliability. For instance, *rep-4* transfers and stores two times more data than *rs-4*. However, we also should notice that the RS technique imposes an overhead due to the encoding work. In the next section we measure and compare the performance overhead of both approaches.

5 Experimental evaluation

This section evaluates the benefits of our proposal both in synthetic settings and for scientific HPC applications.

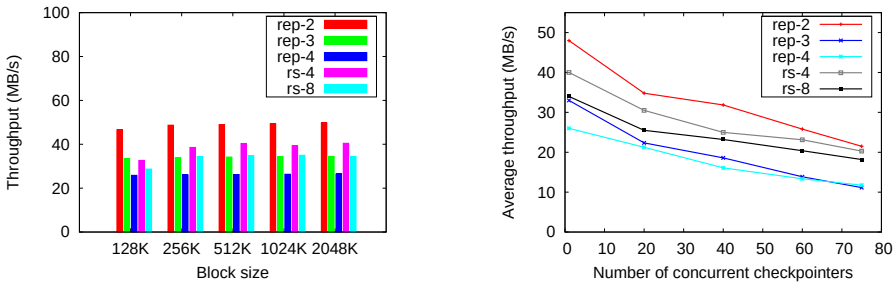
5.1 Experimental setup

The experiments were performed on Grid'5000 [7], an experimental testbed for distributed computing that federates nine sites in France. We used 100 nodes of the griffon cluster from the Nancy site, each of which is equipped with a quadcore Intel Xeon X3440 x86_64 CPU with hardware support for virtualization, local disk storage of 278 GB (access speed $\simeq$55 MB/s using SATA II AHCI driver) and 16 GB of RAM. The nodes are interconnected with Gigabit Ethernet (measured 117.5 MB/s for TCP sockets with MTU = 1500 B with a latency of $\simeq$0.1 ms).

The hypervisor running on all compute nodes is KVM 0.14.0, while the operating system is a recent Debian Sid Linux distribution. For all experiments, a 2 GB raw disk image file based on the same Debian Sid distribution was used to boot the guest operating system of the virtual machine instances that run the user application.

5.2 Synthetic benchmarks

Our first series of experiments evaluates the performance of our approach vs. replication in two synthetic benchmarking scenarios. We compare the same five settings as in Sec-



(a) Single checkpointer, variable chunk size

(b) Increasing nr. of concurrent checkpointers, fixed chunk size (512K)

Fig. 3. Our approach vs. replication: Sustained throughput when checkpointing 512MB worth of data (no computation running in parallel)

tion 4. In all five settings we aim to measure the maximal theoretical performance levels that can be achieved during checkpointing. We chose checkpointing because it is one of the tasks that stress the most the storage. More specifically, we measure the sustained throughput when checkpointing local modifications to the virtual disk that amount to 512MB. We omit running any application in parallel with the checkpointing process in order to better control the experimental setting and eliminate any influence caused by the computation. Instead, the local modifications are simply randomly generated data that is written to the virtual disk before checkpointing.

The first benchmarking scenario evaluates how the chunk size impacts the sustained throughput. To this end, we deploy BlobCR on all available nodes (100) and run a single checkpointing process in each of the five settings using a variable chunk size, ranging from 128K to 2048K. The results are depicted in Figure 3(a). As expected, *rep-2*, *rep-3* and *rep-4* roughly achieve 1/2, 1/3 and 1/4 of the maximal throughput, with a slightly increasing trend as the chunk size gets larger (which is the consequence of fewer messages and thus lower overhead due to latency). This trend is observable for *rs-4* and *rs-8* too, slightly more pronounced due to the larger number of messages (i.e. 4 and 8 respectively per chunk).

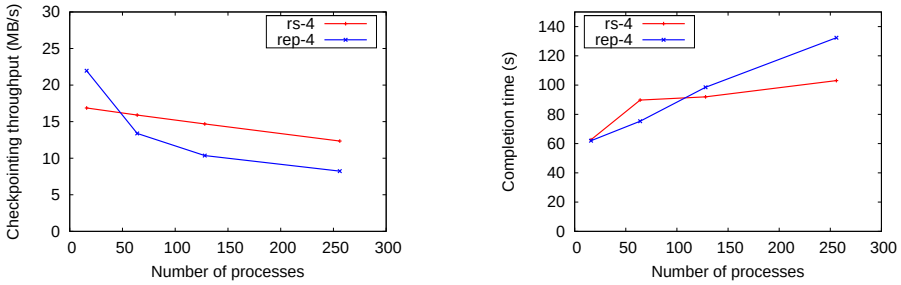
When comparing our approach to replication, the advantage of lower amount of data transfer is clearly visible: for 2048K chunks, *rs-4* achieves a throughput that is 38% higher than *rep-4* while guaranteeing a comparable reliability level. At the same time, it reduces bandwidth and storage space utilization by more than 100% as predicted by our model (See Section 4). Comparing *rs-4* to *rep-2* (which consumes the same amount of bandwidth and storage space), we observe a decrease in throughput of less than 20%. This overhead is a worthwhile investment, considering that *rs-4* increases the reliability level over *rep-2* by several orders of magnitude. Even for small chunk sizes (going as low as 128KB), the advantage of our approach is clearly visible: the throughput achieved by *rs-4* is only 24% lower than *rep-2* and more than 25% higher than *rep-*

4. Finally, we note better scalability for our approach: *rs-8* is already 33% faster than *rep-4*, while increasing the reliability level yet again several orders of magnitude.

So far we have analyzed our approach vs. replication for a single checkpointing process only. Our second benchmarking scenario evaluates how the five settings compare in a highly concurrent setting where an increasing number of checkpointing processes need to save the checkpointing data simultaneously and thus compete for the system bandwidth. We fix the chunk size to 512K and gradually increase the number of checkpointing processes, from one up to 75, while measuring the average sustained throughput. These results are represented in Figure 3(b).

As can be observed, all five approaches suffer a performance degradation with increasing number of checkpointing processes. In case of *rep-2*, the throughput drops from 47 MB/s to little over 20 MB/s. Although starting lower than *rep-2*, both *rs-4* and *rs-8* catch up with *rep-2* under concurrency: the average throughput drops to 20 MB/s and 19 MB/s respectively. This shows that competition for system bandwidth effectively hides the encoding overhead of our approach at larger scales, enabling it to be more scalable: we sustain virtually the same throughput as *rep-2* for the same storage space and bandwidth utilization, albeit at a much higher resilience level. Compared to *rep-3* and *rep-4*, where the throughput drops to little over 10 MB/s, we can observe an even more dramatic improvement than in the case of a single checkpointing process: we sustain a throughput almost 100% higher while keeping the same reliability level (*rs-4*) and even increasing it several levels of magnitude (*rs-8*).

5.3 Real-life application: CM1



(a) Asynchronous checkpointing throughput (higher is better)

(b) Time to finish running the application (lower is better)

Fig. 4. Our approach vs. replication: Performance in the real world (CM1)

In the previous section we have evaluated the throughput of our persistent storage under concurrency for an ideal environment where no application is running. However, in real life the application continues running after requesting a virtual disk snapshot to be taken, while the virtual disk snapshot is persisted in the background. This limits

the amount of resources available to the persistency process: (1) there is less overall available system bandwidth, as the application generates network traffic; (2) there is less available computational power on each compute node, as the application processes run inside the virtual machine and consume CPU.

To illustrate the impact of these limitations in real life, we have chosen *CM1*: a three-dimensional, non-hydrostatic, non-linear, time-dependent numerical model suitable for idealized studies of atmospheric phenomena. This application is used to study small-scale phenomena that occur in the atmosphere of the Earth (such as hurricanes) and is representative of a large class of HPC applications that perform stencil (nearest-neighbor) computations.

We perform a weak scalability experiment that solves an increasingly large problem (constant workload per core), starting from 16 processes up to 256 processes. As input data for CM1, we have chosen a 3D hurricane that is a version of the Bryan and Rotunno simulations [9]. The processes are distributed in pairs of four among virtual machine instances that are allotted 4 cores, such that each process has its own dedicated core. This represents the worst case scenario for our approach, as the block encoding must compete with the application processes. We compare two approaches that offer the same reliability level: *rs-4* and *rep-4*. The checkpointing frequency is fixed such that a single checkpoint is taken throughout the application run-time.

The results are depicted in Figure 4. As expected, the average checkpointing throughput (Figure 4(a)) is smaller than in the ideal case (See Section 5.2) and drops with increasing number of processes. The combined effects of both concurrent checkpointing and application communication quickly saturate the system bandwidth in the case of *rep-4*, which leads to a dramatic drop in average checkpointing throughput from 22 MB/s to 8 MB/s. Our approach on the other hand is more scalable: it presents a drop from 16 MB/s to 12 MB/s. Unlike the ideal case, this time *rs-4* is significantly slower because less computation power is available for encoding. Nevertheless, it is still 50% faster than *rep-4 in the worst case* and has the potential to become even 150% faster if a dedicated core can be spared for the encoding. Taking a look at the completion times (Figure 4(b)), *rs-4* is again much more scalable: the completion time increases from 60s to 100s, which is 35% less than the increase observed in the case of *rep-4* (135s). This advantage of *rs-4* over *rep-4* can be traced back to the twice as lower bandwidth consumption, which effectively increases the bandwidth available to the application.

6 Conclusions

A large class of HPC applications can take advantage of local storage in order to improve performance and scalability while reducing resource utilization, however doing so raises reliability issues. In this paper we have presented a scalable Reed-Solomon based algorithm that provides a high degree of reliability for local storage at the cost of very low computational overhead and using a minimal amount of communication.

We demonstrated the benefits of our approach both theoretically through a performance and resource utilization model, as well as in practice through extensive experiments performed on the G5K testbed. Compared to transitional approaches that rely on replication, we show up to 50% higher throughput and 2x lower bandwidth / storage

space consumption for the same reliability level, which improves overall performance of a real life HPC application (CM1) up to 35%.

Based on these results, we plan to explore in future work the issue of reliability of local storage in greater detail. In particular, we are investigating how to extend our approach to enable high availability of data under concurrent read scenarios: in this context, parity information could be used to avoid contention to the original data. This is important for a large number of HPC applications that need to share the same initial datasets between processes.

Acknowledgments

This work was supported in part by the ANR-10-01-SEGI project, the ANR-JST FP3C project and the Joint Laboratory for Petascale Computing, an INRIA-UIUC initiative. The experiments presented in this paper were carried out using the Grid'5000/ ALADDIN-G5K experimental testbed, an initiative of the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.fr/).

References

1. Tsubame2. http://tsubame.gsic.titech.ac.jp
2. Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/
3. Amazon Simple Storage Service (S3). http://aws.amazon.com/s3/
4. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM 53, 50–58 (April 2010)
5. Bautista-Gomez, L.A., Nukada, A., Maruyama, N., Cappello, F., Matsuoka, S.: Low-overhead diskless checkpoint for hybrid computing systems. In: HiPC '10: Proceedings of the 2010 International Conference on High Performance Computing. pp. 1–10. Goa, India (2010)
6. Bautista-Gomez, L.A., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N., Matsuoka, S.: FTI: high performance fault tolerance interface for hybrid systems. In: SC '11: 24th International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 32:1–32:12. Seattle, USA (2011)
7. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.G., Touche, I.: Grid'5000: A large scale and highly reconfigurable experimental grid testbed. Int. J. High Perform. Comput. Appl. 20, 481–494 (November 2006)
8. Brown, A., Patterson, D.A.: Towards availability benchmarks: a case study of software raid systems. In: ATEC '00: Proceedings of the USENIX Annual Technical Conference. pp. 22:1–22:15. USENIX Association, San Diego, California (2000)
9. Bryan, G.H., Rotunno, R.: The maximum intensity of tropical cyclones in axisymmetric numerical model simulations. Journal of the American Meteorological Society 137, 1770–1789 (2009)
10. Carns, P.H., Ligon, W.B., Ross, R.B., Thakur, R.: PVFS: A parallel file system for Linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference. pp. 317–327. Atlanta, USA (2000)

11. Chen, Z., Dongarra, J.: A scalable checkpoint encoding algorithm for diskless checkpointing. In: HASE '08: Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium. pp. 71–79. IEEE Computer Society, Nanjing, China (2008)
12. Fan, B., Tantisiriroj, W., Xiao, L., Gibson, G.: Diskreduce: Raid for data-intensive scalable computing. In: PDSW '09: Proceedings of the 4th Annual Workshop on Petascale Data Storage. pp. 6–10. ACM, Portland, USA (2009)
13. Gomez, L.A.B., Maruyama, N., Cappello, F., Matsuoka, S.: Distributed diskless checkpoint for large scale systems. In: CCGRID '10: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. pp. 63–72. CCGRID '10, IEEE Computer Society, Melbourne, Australia (2010)
14. Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44, 35–40 (April 2010)
15. da Lu, C.: Scalable Diskless Checkpointing for Large Parallel Systems. Ph.D. thesis, Univ. of Illinois at Urbana-Champaign (2005)
16. Moody, A., Bronevetsky, G., Mohror, K., Supinski, B.R.d.: Design, modeling, and evaluation of a scalable multi-level checkpointing system. In: SC '10: Proceedings of the 23rd International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–11. IEEE Computer Society, New Orleans, USA (2010)
17. Nadgowda, S.J., Sion, R.: Cloud Performance Benchmark Series: Amazon EBS, S3, and EC2 Instance Local Storage. Tech. rep., Stony Brook University (2010)
18. Nicolae, B., Antoniu, G., Bougé, L., Moise, D., Carpen-Amarie, A.: BlobSeer: Next-generation data management for large scale infrastructures. J. Parallel Distrib. Comput. 71, 169–184 (February 2011)
19. Nicolae, B., Cappello, F.: BlobCR: Efficient Checkpoint-Restart for HPC Applications on IaaS Clouds using Virtual Disk Image Snapshots. In: SC '11: 24th International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 34:1–34:12. Seattle, USA (2011)
20. Plank, J., Xu, L.: Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In: Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on. pp. 173 –180 (july 2006)
21. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8(2), 300–304 (1960)
22. Schmuck, F., Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies. Monterey, USA (2002)
23. Shvachko, K., Huang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: MSST '10: 26th IEEE Symposium on Massive Storage Systems and Technologies. pp. 1–10 (2010)
24. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: SoCC '10: Proceedings of the 1st ACM Symposium on Cloud Computing. pp. 193–204. Indianapolis, USA (2010)
25. Weatherspoon, H., Kubiatowicz, J.: Erasure coding vs. replication: A quantitative comparison. In: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems. pp. 328–338. Springer-Verlag, London, UK (2002)
26. Zhang, Z., Deshpande, A., Ma, X., Thereska, E., Narayanan, D.: Does erasure coding have a role to play in my data center? Tech. Rep. MSR-TR-2010-52, Microsoft Research (2010)